# Why do structures get tag names even if there is a typedef?

devblogs.microsoft.com/oldnewthing/20080327-00

March 27, 2008

Raymond Chen

As we noted last time, structure tags are different from the typedef name as a historical artifact of earlier versions of the C language. But what about just leaving out the name entirely?

```
typedef struct {
 ...
} XYZ;
```

One problem with this approach is that it becomes impossible to make a forward reference to this structure because it has no name. For example, if you wanted to write a prototype for a function that took one of these structures, and you could not be sure that the header file defining the `XYZ` type definition has already been included, you can still refer to the structure by its tag name.

```
// in header file A
typedef struct tagXYZ {
 ...
} XYZ;
// in header file B
BOOL InitializeFromXYZ(const struct tagXYZ *pxyz);
```

The two header files can be included in either order because header file B uses a forward reference to the `XYZ` structure. Naturally, you would hope that people would include header file A before header file B, but there can be cases where it is not practical. (For example, header file A may contain definitions that conflict with something else that the program needs, or header file A may change its behavior based on what has already been `#define`'d, and you don't want to include it before the application has a chance to set up those `#define`s.)

But a more important reason to avoid anonymous types is that it creates problems for MIDL.

Okay, it doesn't actually create problems for MIDL. MIDL handles it just fine, but the way MIDL handles it *creates problems for you*, for when you create an anonymous type in MIDL, such as an anonymous structure above, or an anonymous enumeration like this:

```
typedef enum { ... } XYZ;
```

MIDL *auto-generates a name for you*. For example, the above enumeration might end up in the generated header file as

```
typedef enum __MIDL___MIDL_itf_scratch_0000_0001
{
    ...
} XYZ;
```

The kicker is that the auto-generated name changes if you change the IDL file. And since typedefs are just shorthand for the underlying type (rather than a type in and of themselves), the name saved in the PDB is the unwieldy `__MIDL___MIDL_itf_scratch_0000_0001`. Try typing that into the debugger, yuck.

Furthermore, having the name change from build to build means that you have to make sure code libraries are all built from exactly the same header file versions, even if the changes are ostensibly compatible. For example, suppose you compile a library with a particular version of the header file, and then you add a structure to the MIDL file which has no effect on the functions and structures that the library used. But still, since you changed the MIDL file, this changes the auto-generated symbol names. Now you compile a program with the new header file and link against the library. Result: A whole bunch of errors, because the library, say, exports a function that expects its first parameter to be a `__MIDL___MIDL_itf_scratch_0000_0001` (because the library was built from the older MIDL-generated header file), but your program imports a function that expects its first parameter to be a `__MIDL___MIDL_itf_scratch_0001_0002` (because you compiled with the newer MIDL-generated header file).

What's more, when you update the header file, your source control system will recognize hundreds of changes, since the MIDL compiler generated a whole different set of names which no longer match the names from the previous version of the header file, even though you didn't change the structure! This isn't fatal, but it makes digging through source code history more of an ordeal since the "real changes" are buried amidst hundreds of lines of meaningless changes.

Now, this particular rule of thumb is not universally adhered-to in Windows header files, in large part, I believe, simple because people aren't aware of the potential for mischief. But maybe now that I wrote them up, people might start paying closer attention.

Raymond Chen

**Follow**